International Journal of Engineering Research and Development
e-ISSN : 2278-067X, p-ISSN : 2278-800X, www.ijerd.com
Volume 2, Issue 5 (July 2012), PP. 37-41

Implementation of a Turbo Encoder and Turbo Decoder on DSP Processor-TMS320C6713

K. N. V. Khasim[3], Ch. Ravi Kumar[1], Dr. K. Padma Raju[2], V. Glory[4]
[1,3,4]*Sir.C.R.R.Engineering College, Eluru, AP, India*
[2]*JNTU College of Engineering Kakinada, AP, India*

***Abstract*—Turbo codes are used for error protection, especially in wireless systems (e.g., in the cdma2000 standard) and in joint source-channel coding. An example of joint source-channel coding is the use of turbo codes in conjunction with JPEG2000 for image communication over noisy channels.**

A turbo encoder consists of two recursive systematic convolutional component encoders connected in parallel and separated by a random interleaver. Different iterative decoding algorithms like the Iterative MAP, the Iterative Log-Map and the Iterative Soft Output Viterbi Algorithm (SOVA) are used to decode the turbo codes. The complexity of the iterative SOVA is much lower than the MAP and the Log-MAP algorithms.

In this project, a turbo encoder and SOVA-based turbo decoder in real-time software are implemented on a TMS320C6713 digital signal processor (DSP).

Keywords*—DSP Processor-TMS320C6713, Turbo codes, RSC Encoder, SOVA Decoder

I. INTRODUCTION

In a digital wireless communication system, the purpose of the channel code is to add redundancy to the binary data stream to combat the effect of signal degradation in the channel. Ideally, channel codes should meet the following requirements:

1) Channel codes should be high rate to maximize data throughput.
2) Channel codes should have good Bit Error Rate (BER) performance at the desired Signal-to-Noise Ratio(s) (SNR) to minimize the energy needed for transmission.
3) Channel codes should have low encoder/decoder complexity to limit the size and cost of the transceivers.
4) Channel codes should introduce only minimal delays, especially in voice transmission, so that no degradation in signal quality is detectable.

These requirements are very difficult to obtain simultaneously; excellent performance in one requirement usually comes at the price of reduced performance in another. However, for cellular voice and data communication, it is desirable that all these requirements be met, which makes cellular communication systems very difficult to design.

In 1993, *Claude Berrou et. al* introduced a class of parallel concatenated convolutional codes, known as Turbo Codes [1]. They claimed a BER performance of 10-5 at an SNR of 0.7 dB for a rate 1/2 binary code. This performance, which was only 0.7 dB from the Shannon limit1, was a large improvement over other coding methods at the time. Initially, Turbo Codes were received by the coding community with a great deal of skepticism, until other researchers were able to reproduce their results.

II. TURBO CODES

A turbo encoder consists of two "recursive systematic convolutional encoders, connected in parallel. The input to the second encoder is an interleaved version of the input to the first encoder. This structure is called parallel because the input to both of the encoders is the same set of bits, rather than the output of one being the input to the other, as in serial concatenated codes. Iterative SOVA based Turbo Decoder [1] a turbo encoder are two recursive systematic convolutional encoders. The are called "systematic" because one of the outputs of the encoders is the input itself, whereas the property "recursive" comes due to the presence of a feedback loop in the encoders. A rate 1=3 turbo encoder is shown in Fig. 1. The generator matrix of a rate 1=2 component code can be represented as

$$G(D) = \cdot 1\ g1(D)g0(D) ......(1)$$

Where D represents the delay and $g0(D)$ and $g1(D)$ are the feedback and feed forward polynomials respectively. The degree of these polynomials is n, which depends on the number of delays in the convolutional encoders. As shown in Fig. 1, the same set of input bits is encoded twice, once by each component encoder. The input to the first encoder is the original input sequence x, whereas the input to the second encoder is the interleaved version of x, denoted by ex. The first output of the turbo encoder, which is the first output of the first encoder as well, is the original input sequence itself, denoted by $y0$. The second output of the first encoder is the parity information $y1$. For the second encoder we are only concerned with the parity output $y2$ and not with the systematic bits. Hence the three outputs $y0$, $y1$ and $y2$ are multiplexed to form the resultant output of the turbo encoder. Hence the resulting rate of this turbo encoder is 1=3.

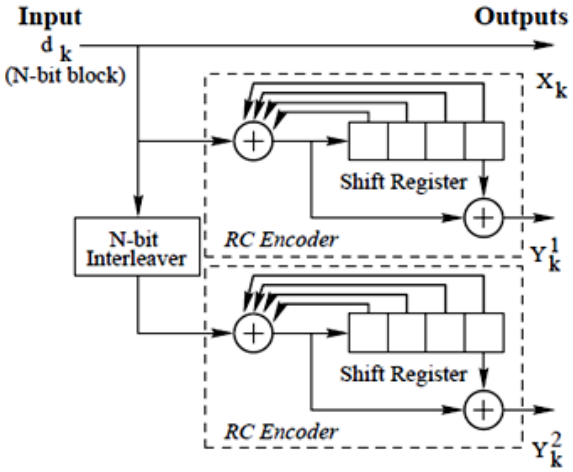

Figure.1. **Rate 1/3 Turbo Code Encoder**

Usually the systematic output of the second RSC encoder is omitted to increase the code rate. Figure.1 shows the Turbo Code encoder. Due to the interleaver between the RSC encoders and the nature of the decoding operation, the Turbo Code encoder must operate on the input data stream in N-bit blocks. (Therefore, Turbo Codes are indeed linear block codes.) Although the interleaver appears to play only a minor role in the encoder, in actuality the interleaver is a very important component. An estimate of the BER performance of a Turbo Code can be made if the codeword weights or code geometry is known. Intuitively, we would like to avoid pairing low-weight code words from the upper RSC encoder with low-weight words from $\{Y_k^2\}$. Many of these undesirable low-weight pairings can be avoided by properly designing the interleaver. Since the RSC encoders have infinite impulse response, the input sequence consisting of a single one will produce a high-weight codeword. However, two suitably placed 1's in the input sequence can drive the RSC encoder out of the zero state and back again within a very short span. If the interleaver maps these sequences to similar sequences, the over-all codeword weight will be low.

For moderate SNR values, the weight distribution of the first several low-weight code words is required. For small interleaver sizes (*N*), the complete weight distribution can be determined by an exhaustive search using a computer. Calculating (or even estimating) the weight distribution for large N is a very difficult problem. Once the weight distribution is known, an estimate of the bit error rate can be made by using the union bound.

Upper Bound on BER Performance An upper bound on the BER performance of an (n; k) systematic block code is given by the union bound,

$$P_b \leq \sum_{w=1}^{k} \sum_{d=d_{min}}^{n} a(w,d) \frac{w}{n} \frac{1}{2} \operatorname{erfc}\left(\sqrt{d \frac{k}{n} \frac{E_b}{N_0}}\right)$$

where, *Eb/N0* is the signal-to-noise ratio, *w* is the Hamming weight of the input block, d is the Hamming weight of the codeword, and *a(w; d)*, which is known as the weight enumeration function, is the number of such codewords.

For simple block codes, *a (w; d)* can be obtained by an exhaustive search. However, for long block lengths, obtaining *a (w; d)* can be very difficult. *Benedetto* and *Montorsi* present a method for obtaining the weight enumeration function of a Turbo Code. An alternative method for obtaining the weight enumeration function of the constituent RSC codes is, in the trellis diagram of an RSC code, each path in the trellis represents a transition from a previous state m', to a current state m with an input bit X_k and a parity bit Y_k.

We define the single stage trellis transition matrix as follows:

$$T_{m',m}(W,Z) = [a_{m',m}(w,j) W^w Z^j]$$

where *w* is the Hamming weight of the input bit X_k, j is the Hamming weight of the parity bit Y_k, and $a_{m',m}(w; j)$ is the number of transitions in the trellis from state m_0 to state *m* with input weight *w* and parity bit weight *j*. Note that the Hamming weight of the complete codeword is *w + j*. The weight distribution function for *N* stages of the trellis (corresponding to an *N* bit block) is obtained by calculating $T_{m',m}^{N+M}(W,Z)$, where *M* is the number of memory elements (assuming that *M* tail bits are added to terminate the trellis). Each entry in $T_{m',m}^{N+M}(W,Z)$ gives the input redundant weight enumeration function [17] for all trellis paths that start at state m' and end at state *m* after *N +M* transitions. The entry which is of most interest is $T_{0,0}^{N+M}(W,Z)$, which is the input redundant weight enumeration function of all the paths that start and end at the all-zero state. Since the upper bound on the BER performance is dominated by the lower weight codewords, the calculation of

$T_{m',m}^{N+M}(W,Z)$ can be simplified by only enumerating codewords with input and parity Hamming weights below certain thresholds.

For parallel concatenated block codes, such as Turbo Codes, two linear systematic codes C1 and C2 are linked by an interleaver. In order to obtain the weight enumeration function of such a parallel code, the calculation must take into account each constituent code and the interleaver structure. Since this calculation becomes impractical to perform for even small block lengths, Benedetto and Montorsi introduced an abstract interleaver which they called a uniform interleaver. A uniform interleaver of length *k* is a probabilistic device which maps a given input word of weight *w* into all the distinct $\binom{k}{w}$ permutations with equal probability $1/\binom{k}{w}$ The definition of a random interleaver results in a weight enumeration function for the second code which is independent of the first code. Therefore, the weight enumeration function for the parallel code can be calculated as follows:

$$A_w^{C_p}(Z) = A_w^{C_1} A_w^{C_2} / \binom{k}{w},$$

where $A_w^{C_1}, A_w^{C_2}$ are the weight enumeration functions of the constituent codes. In the previous equation, $T_{0,0}^{k}(W,Z)$ can be converted into weight enumeration functions and used for $A_w^{C_1}$ and $A_w^{C_2}$. Finally, the upper bound on the BER performance is given by calculating P_b equation using $A_w^{C_p}(Z)$. The upper bound on the BER performance of Turbo Codes are shown in Figure 2 and Figure 3 for various block lengths (N) and memory sizes (M). Specifically, the bounds *for M = 3, G = (15; 13) and M = 4, G = (37; 21) and G = (23; 35)* RSC encoders are shown.

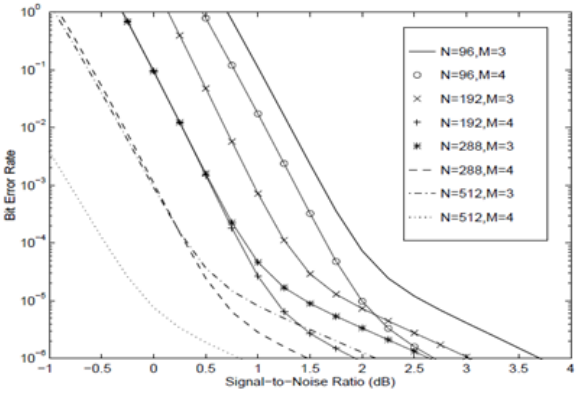

Figure.2: **Upper Bounds on BER Performance of Turbo Codes for M = 3, G = (15; 13) and M = 4, G = (23; 35).**

The curves in Figure.2 and Figure.3 have two major segments with greatly different slopes. The shallow sloping portions of the curves, for bit error rates below 10-5, are dominated by a few very low weight codewords. For bit error rates greater than 10^{-5}, the curves show the impact of the first several low-weight codewords. Figure 2.5 show the upper bounds on the BER performance of Turbo Codes with *M = 3* and *M = 4* memory element RSC encoders. As the block length increases, the difference between these two codes also increase. For example, a BER of 10^{-4} can be obtained for *M = 3;N = 288* or *M = 4;N = 192*. Therefore, there is a trade-off between block size (and thus interleaver delay) and memory size (which corresponds to decoder complexity).

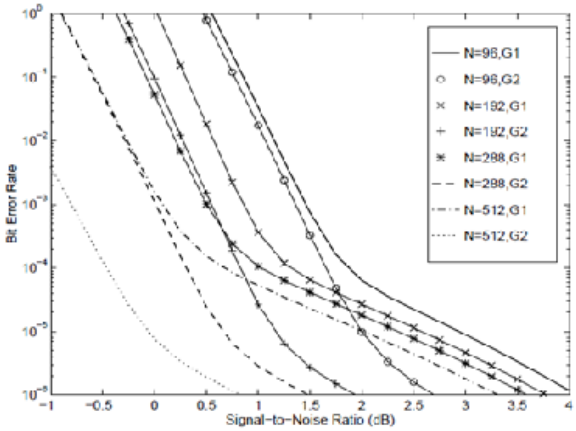

Figure.3. **Upper Bounds on BER Performance of Turbo Codes for M = 4, G1 =(37; 21) and G2 = (23; 35).**

Figure 4.5 show the upper bounds on the BER performance of two Turbo Codes with different generators. Since the feedback polynomial $(37)_8$ does not produce a maximum length sequence, the upper bound for both the steep and shallow portions of the curves are inferior to the second generator with feedback polynomial $(23)_8$ which does produce a maximum length sequence.

TURBO DECODING

The decoding of the turbo codes can be performed using the Maximum a Posteriori (MAP) algorithm or the Maximum Likelihood (ML) algorithm based on the overall trellis of the code. However, these methods can be implemented only for short interleavers and are too complex for medium and long interleavers. One of the important practical features of turbo codes is the use of a simple suboptimum algorithm for decoding. One important reason why simple MAP decoding is not practical is that the overall trellis for the turbo codes is time varying and the number of states grows exponentially with the size of the interleaver in the turbo codes. Hence the MAP algorithm can only be used to decode in the case of very short interleavers. Due to these reasons, iterative decoding algorithms are used to decode the turbo codes. Different iterative decoding algorithms like the Iterative MAP [1], the Iterative Log-Map [1] and the Iterative Soft Output Viterbi Algorithm (SOVA) [1] are used to decode the turbo codes. In this report we only present iterative SOVA because it is shown in [1, 6] that the complexity of the iterative SOVA is much lower than the MAP and the Log-MAP algorithms. The block diagram of the iterative SOVA is shown in Fig.4

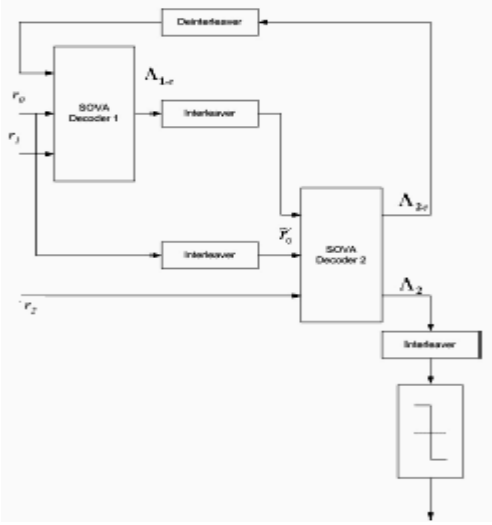

Figure.4 **SOVA based Decoder**

The input to the first decoder is the received bits corresponding to the actual information (systematic) bits, and the output of the first component encoder, represented by *r*0 and *r*1 respectively. This first decoder generates a soft estimate of the output and the extrinsic information ^1*e*. This extrinsic information is passed through the same interleaver as the one used in the encoder and is passed to the second decoder, to be used as an estimate of the priori probability by the second decoder. The inputs to the second decoder are the received bits corresponding to the output of the second encoder, represented by *r*2, and *r*0 passed through the same interleaver as used at the encoder, represented by e *r*0. The second decoder also generates a soft decision ^2 and the extrinsic information ^2*e*. This extrinsic information is then passed back to the first encoder after deinterleaving, as an a priori estimate for the next decoding iteration. The decoder generates a hard decision after the desired number of iterations *I* are performed and then deinterleaves it and passes it to the output.

III. DSP SOFTWARE IMPLEMENTATION

We implement the channel coding subsystem on a TMS320C6713 DSP. Our goal is to optimize the turbo encoder and SOVA-based turbo decoder for computation time. It will be critical to have all code and critical data segments reside on-chip. Since large blocks of data are needed to be stored efficiently in the internal memory for every subsequent iteration, limited on-chip data memory size of 64KB imposes a challenging task on the optimization of highly complex encoder and iterative SOVA decoder.

We first wrote all of the modules of the channel coding system in C and then compiled them on the TMS320C6713DSP using Code Composer version 1.0. Ultimately, we implemented the encoder, puncture block, and BPSK modulator fixed-point assembly. We realized the SOVA-based decoder and the insert zeros block in floating-point assembly. The code is optimized with respect to execution time and memory usage. Results show that for the encoder, a reduction of approximately 63 times in the execution time is achieved by writing the routine in assembly and using loop unrolling as compared to only memory optimization. Similarly, for the decoder, a reduction of approximately two times in the execution time is achieved. Using memory optimization, execution time was reduced approximately 6 and 1.5 times for the puncture and the insert zeros blocks, respectively. The assembly generated by the code composer after memory optimization for the puncture and the insert zeros block was optimized, and no further optimization was achieved by writing these routines in assembly.

IV. SIMULATION

In order to validate the DSP implementation, the entire system is simulated over a Rayleigh fading channel. The performance of this unequal error protection is also compared to that of the uniform error protection case, in which the same rate turbo code is used to protect the entire bit stream. Rate 2=3 (66.67% transmission efficiency) turbo code is used for the uniform error protection, whereas the equivalent rate for the unequal protection is 3=4 (75% transmission efficiency At a BER of 10¡2, 1 dB of coding gain is achieved for uniform rate puncturing as compared to the variable rate puncturing. It provides larger number of redundant bits and hence offers more error protection.

The greater protection to the more important bits and truncating, results in discarding the less protected, less important bits that contain more errors. Truncating the uniformly protected bits tream does not reduce the BER significantly because the error is uniformly distributed over the entire bit stream.

V. CONCLUSION

In this paper, we presented a real time implementation of the iterative soft output viterbi algorithm based punctured turbo encoder/decoder over a TMS320C6x DSP processor. A speedup of 162x for the encoder and 11.7x for the decoder is achieved over the level three C compiler optimization. All the modules in the system are modeled using computational dataflow models. This is the first publicly available implementation of a SOVA-based turbo decoder on a C6700 DSP processor.

REFERENCES

[1]. Z. Wang and A. C. Bovik, "Embedded Foveation Image Coding *IEEE Transactions on Image Processing*, vol. 10, pp. 1397–1410, Oct. 2001.

[2]. A. Ushirokawa, T. Okamura, N. Kamiya, and B. Vucetic, "Principles of turbo codes and their applications to mobile communciations," *IEICE Transactions on Fundamentals*, vol. E81-A, pp. 1320–1329, July 1998.

[3]. B. Vucetic and J. Yuan, *Turbo Codes*. Kluwer Academic Publishers, 2000.

[4]. C. Berrou and A. Glavieux, "Near optimum error correcting coding and decoding: turbo-codes," *IEEE Transactions on Communications*, vol. 44, pp. 1261 –1271, Oct. 1996.

[5]. B. Banister, B. Belzer, and T. Fischer, "Robust image transmission using JPEG2000 and turbo-codes," *IEEE Signal Processing Letters*, vol. 9, pp. 117–119, Apr. 2002.